CSCE 420 Individual Project Specification
Due: 11:59 P.M. Tuesday, December 5, 2017

The project is to design and train a neural network to give a better estimate of the number of remaining moves for a state of the 15-puzzle than "Manhattan distance" gives. For example, the puzzle state

	12	11	13
15	14	10	9
3	7	6	2
4	8	5	1

actually takes 80 moves to reach the standard goal state

1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

but the Manhattan distance estimates only 56. Since this is not an overestimate, Manhattan distance can be used as the "**h** heuristic" in the A* algorithm, but the performance of A* would be much better with a better **h**.

You will be supplied with 29 files containing all the 15-puzzle states requiring exactly **n** moves to reach the standard goal state for **n** from 0 through 28. (**n** is the minimum, that is, each puzzle cannot be solved in less than **n** moves.) The files are named **0states.bin** through **28states.bin**. Note: Some of the files are very large, but since they have shared read access you do not have to download them to your account. The neural network is then trained to produce an output of **n** for each of the states in the corresponding <**n**>states.bin file, for all 29 files.

Since demultiplexing the input could require several additional neuron layers, you should convert the puzzle state read in to 240 bits as follows:

- Open the file in binary mode
- Read 8 bytes as a 64-bit integer

Warning: Check the endian-ness of your data. Here's how: the single state in 0states.bin should read in as equal to 0xf23456789abcdef0. If it doesn't, reverse the order of the 8 bytes to get this value.

- Ignore the first 4 (i.e., most significant) bits
- Decode the remaining 60 bits as 15 fields of 4 bits each representing the value of the tile (0 for the blank) in each position of the puzzle except [0][0], stored row-wise as in C++. Thus 0xf23456789abcdef0 means

?	2	3	4
5	6	7	8
9	10	11	12

13	14	15	

Note that for this project we don't have to deduce what tile is in [0][0] since it is completely determined by the other 15 positions and would not add any information (negative entropy).

- Now express each 4-bit value as 16 bits with one bit corresponding to each value:
  0 is expressed as 0b0000'0000'0000'0001 (the 1 is in the 2^0 position)
  1 is expressed as 0b0000'0000'0000'0010 (the 1 is in the 2^1 position)
  2 is expressed as 0b0000'0000'0000'0100 (the 1 is in the 2^2 position)
  …
  15 is expressed as 0b1000'0000'0000'0000 (the 1 is in the 2^{15} position)
- So 0xf23456789abcdef0 becomes
  0b0000'0000'0000'0100'0000'0000'0000'1000…0000'0000'0000'0001 which are input to the 240 input-layer neurons.

Similarly, the output layer has 29 neurons, one for each value 0 through 28. Since state 0xf23456789abcdef0 has 0 moves remaining, the 29 outputs should be 0b0'0000'0000'0000'0000'0000'0000'0001 (the 1 is in the 2^0 position). Until the neural network is trained it will produce values between 0 and 1 as well as 0 and 1, so discretize the output by putting a 1 in the highest output and zeroes elsewhere.

Since we are trying to get a more accurate **h** output, we will evaluate accuracy by how much of the possible improvement was obtained. For the 80-move puzzle above, the Manhattan distance is 56, so if the neural network learns to produce 68, that is 50% of the possible improvement, and if it produces 74 that is 75% of the possible improvement. (When the Manhattan distance is correct, *e.g.*, for the 0-move state, then no improvement is possible.) The overall "quality" of the neural network is the average percentage of possible improvement for states which could be improved.

(Note that this is not a good use of neural networks; the point of the project is to experiment with and understand neural networks without having to first learn another area like computer vision!)

You must use 240 input neurons and 29 output neurons (to match the data), but you may experiment with one or more hidden layers of various sizes. (You must have at least one hidden layer.) A common approach is to halve the number of neurons; for example, 240-120-60-29.

You may also experiment with different values of alpha (the training rate), or even a training schedule. For example, if you run **k** training cycles, alpha could increase as 1/**k**, 2/**k**, …, **k**/**k**.

Use the logistic sigmoid function for the output layer. However, since it is essentially zero for very negative values, that could prevent learning from the previous layer, so use hyperbolic tangent for all other layers, including the input layer. In both cases the derivative (needed for back-propagation) is easy to compute:

- For the logistic sigmoid function, $g'(x) = g(x) \cdot (1 - g(x))$
- For the hyperbolic tangent, $g'(x) = 1 - (g(x))^2$

Instead of computing g(x) every time, precompute a table of, say, 10000 values of x from -8 to 8 and then just look up the nearest table value. Otherwise most of the back-propagation time will be spent evaluating g and g'!

The simplest approach is to code the back-propagation neural network training algorithm pseudocode in Chapter 18 of the textbook. You are free to use other packages (Google TensorFlow, Microsoft CNTK, Berkeley Caffe, Keras, Mathematica, etc.) so long as you give credit in your report and in comments in the code you write to interface with a package. **Be sure to use compute.cse.tamu.edu and not build.tamu.edu!**

Your program must be submitted both to CSNET and also on a CD or DVD. The project report (described below) should be submitted on paper to your TA, along with your CD or DVD. Write a report according to the outline below.

REPORT OUTLINE

The project report must be printed on a laser printer. The report should include the following sections:

1. Statement of the problem, significance, etc.
2. Restrictions and limitations
3. Explanation of your approach (analysis and experiments to choose how many layers, how many neurons in each layer, learning rate and schedule, number of back-propagation iterations, etc.)
4. Sample run (screen shots)
5. Results and analysis
6. Conclusions - What did you show? What did you learn?
7. Future research (how your program could be improved or extended)
8. Instructions on how to run your program
9. Listing of the COMMENTED program
10. Bibliography - references used, if any